NPS52-89-051

NAVAL POSTGRADUATE SCHOOL

Monterey, California

FALTER -- A Fault Annotation Tool

Timothy J. Shimeall

September 1989

Prepared for:

Naval Postgraduate School
Monterey, California 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. W. West, Jr.
Superintendent

Harrison Shull
Provost

This report was prepared in conjunction with research funded by the Naval Postgraduate School Research Council.

UNCLASSIFIED

IRITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
DECLASSIFICATION/DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S): NPS52-89-051	5. MONITORING ORGANIZATION REPORT NUMBER(S)
NAME OF PERFORMING ORGANIZATION: mputer Science Dept. / val Postgraduate School; 6b. OFFICE SYMBOL (if applicable): 52	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
ADDRESS (City, State, and ZIP Code): onterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943
NAME OF FUNDING/SPONSORING ORGANIZATION: val Postgraduate School; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: O&MN direct funding
ADDRESS (City, State, and ZIP Code): onterey, CA 93943	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

TITLE (Include Security Classification)
ALTER -- A Fault Annotation Tool(U)

PERSONAL AUTHOR(S)
HIMEALL, Timothy J.

a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Progress	FROM 9/88 TO 9/89	89 September 26	12

SUPPLEMENTARY NOTATION

COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Software Testing, Fault, Failure Region

ABSTRACT (Continue on reverse if necessary and identify by block number)

ALTER is a program that supports the process of determining the effect of a program defect on the local ogram state. FALTER also provides the capability of recording the effect by annotation of the program ntrol flow graph. FALTER is one of a series of four tools that work in an integrated fashion to analyze ascal programs to determine the failure regions associated with identified faults in the programs. The notated control flow graph produced by FALTER will used as input by the program SPACER, and shall be stomized for such usage. The users may access REACHER, FALTER and SPACER through a screen-ented user interface called VIEWER. Beyond the failure region analysis FALTER may be useful in search that examines the distribution of faults in program source code, and in efforts that examine the roneous transformations induced by faults.

DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED
a. NAME OF RESPONSIBLE INDIVIDUAL: Shimeall, Timothy J.	22b. TELEPHONE (Include Area Code): (408) 646-2509 / 22c. OFFICE SYMBOL: 52Sm

FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Environment for Failure Region Analysis:

FALTER -- A Fault Annotation Tool

Timothy Shimeall

Computer Science Department (Code 52Sm)
Naval Postgraduate School
Monterey CA 93943

25 September 1989

Table of Contents

List of Tables

List of Figures

1.0 Introduction: FALTER -- A Fault Annotation Tool

FALTER is a program that supports the process of determining the effect of a program defect on the local program state. FALTER also provides the capability of recording the effect by annotation of the program control flow graph (generated by REACHER).

In at least the initial release of FALTER, the onus of derivation of the fault conditions will fall on the user. It is therefore important that the user of FALTER be a knowledgeable researcher, with experience in faults and their description.

FALTER is one of a series of four tools that work in an integrated fashion to analyze Pascal programs to determine the failure regions associated with identified faults in the programs. The annotated control flow graph produced by FALTER will used as input by the program SPACER, and shall be customized for such usage. The users may access REACHER, FALTER and SPACER through a screen-oriented user interface called VIEWER. Figure 1 provides a context diagram for this use of FALTER.

Beyond the failure region analysis FALTER may be useful in research that examines the distribution of faults in program source code, and in efforts that examine the erroneous transformations induced by faults.

FALTER shall be written in C for use under UNIX 4.3 BSD. Future versions may be transported to other operating systems and versions of BSD. Future versions may also be constructed that deal with other input languages, in particular Ada (trademark, DoD AJPO).

This document contains all requirements for FALTER. Section 2 is a description of the input and output data for FALTER. Two forms of description are used to describe the data. Data entered or generated in a specific format is described using a BNF-style

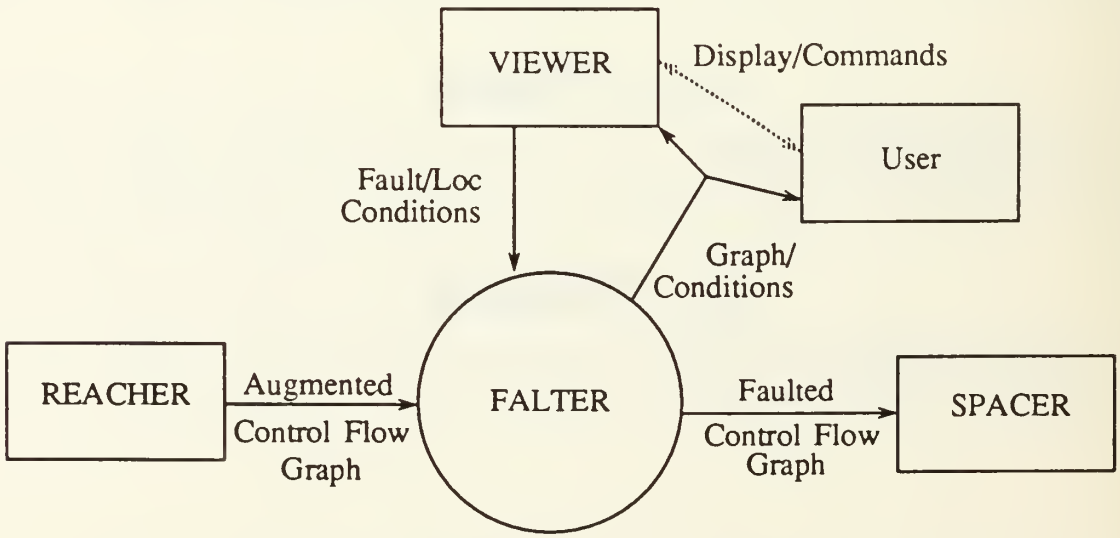

Figure 1: Context Diagram for REACHER

description, with non-terminals in *italics*, terminals in **bold**, explanations of non-terminals in normal print and alternatives definitions are indicated by the vertical bar 'I'. Data entered or generated with specific components of information are described in a record-style format.

Section 3 is a list of all of the functional requirements, including a description of the response to each possible program input. Terms found in the Glossary are !delimited by exclamation points!. /Input variables/ are delimited by slashes. //Output variables// or portions thereof are delimited by doubled slashes. $Symbolic Value References$ are delimited by dollar signs. In this section, the verb ''shall'' is used to indicate required behaviors for FALTER. The verbs ''will'' or ''is'' is used to indicate necessary or desirable actions that occur beyond the control of FALTER (e.g., user actions). The verb ''may'' is used to indicate optional or alternative actions.

Section 4 identifies all acceptable subsets and foreseen supersets(extensions) to the basic functionality described in sections 2 and 3.

Section 5 identifies the foreseen undesired events that may occur during FALTER's execution and describes responses to these undesired events. Omitted from this section are events that may occur during FALTER's execution, but that FALTER cannot respond to. Duplicatively included in this section are all error messages produced by FALTER and the conditions under which FALTER will generate these messages.

Section 6 is a glossary of defined terms used in this document. In the text of this document, each defined term appears delimited by exclamation points. These defined terms may be looked upon as text macros, and these terms should be read in context.

2.0 Data Descriptions

Input

1. Augmented Control-Flow Graph (/ACFGHDR/, /ABKHDR/, /ACFG/)
   (See REACHER Requirements Document)

2. Fault Conditions (/FaultCond/)
   Most faults affect only selected portions of the local software state, and the affect produces an erroneous state only under specialized conditions. Thus, the fault is an implication:

   fault-cond ::= (*selection-cond*) **and** (*error-cond*) **->** (*error-transform*)

   where

selection-cond	is a boolean expression selecting the affected portion of the local state.
error-cond	is a boolean expression selecting the conditions under which the error-transform occurs.
error-transform	is a boolean expression describing the logical transformation of the system state.

3. Location Conditions (/LocCond/)
   Most faults may be attributed to specific portions of the program source code. However, some faults may be more distributed in the source. As such, it is useful to provide for a grammar to describe the location of a fault.

   loc-cond ::= *Integer* | *Integer* .. *Integer* | *Integer* .. *Integer* **given** *loc-selection*

   where

Integer	is a normal Pascal integer (non-negative)
loc-selection	is a Pascal boolean expression

Output

1. Faulted Control-Flow Graph (//FCFGINFO//)
   (Similar to /ACFGHDR/, /ABKHDR/ referenced above)
   The format of this output will be specialized to be compatible with SPACER's expected input.
   1. FCFG Header Info (//FCFGHDR//)

Field	Acronym	Value
Number of Graphs	//FHLEN//	Integer
Graph Data	//FHPROCS//	List of //FBKHDR//
Program Name	//FHPGRM//	String

   2. FCFG Block Header Info (//FBKHDR//)

Field	Acronym	Value
Block Name	//FBKNAME//	String
Number of Return Locations	//FBKNUMRET//	Integer
Return Locations	//FBKRET//	List of /ACFG/
Entry Conditions	//FBKREACH//	//ReachCond//
Block Nodes	//FBKGRPH//	/ACFG/
Number of Subsidiary Blocks	//FBKNSUBS//	Integer
Subsidiary Blocks	//FBKSUBS//	List of //FBKHDR//
Declaration Text	//FBKDECL//	String
Number of Faults	//FBKFNUM//	Integer
Fault Starting Points	//FBKFLOC//	list of /ACFG/
Fault Conditions	//FBKFCON//	list of //Conditional//
Fault Information	//FBKFDATA//	list of //FaultInfo//

   where //Conditional// is a Pascal Boolean expression, and a new structure //FaultInfo// has the following fields:

Field	Acronym	Contents
Fault Identification	//FID//	String
Fault Description	//FDESC//	String
Violated Specification Portion	//FVIOL//	String
Fault Type	//FTYPE//	//FaultClass//
Fault Location	//FLOC//	/LocCond/
Fault Implication	//FIMP//	/FaultCond/

   where //FaultClass// is the set $Overrestrict$, $LoopCond$, $Calc$, $Inital$, Sub, $NoCheck$, $Branch$, $NoBranch$, $NoThread$, $NoReq$, $Order$, $Reverse$, $Data$
2. Graph/Condition Prompts (//GCPrompt//)

Field	Acronym	Contents
Graph Location	//GCLoc//	/ACFG/
Graph Statement Text	//GText//	String
Graph Statement Comments	//GComm//	String
Graph Error Conditions	//GErr//	//Conditional//

3.0 Functional Requirements

3.1 Overview

FALTER prompts the user for the program section where the identified fault first affects the execution (or equivalently, the procedure or function in which the program defect may be corrected). Starting with the first statement of the routine, FALTER steps through statement by statement, constructing a local state in a user-supervised manner. At the point where the fault is identified, FALTER prompts the user with m each section of the local state and requests transformations caused by the fault on that portion of the local state. When all portions of the local state are dealt with, FALTER records the information in the //FCFG// and exits.

3.2 Initial Processing

On program initialization, FALTER shall expect the name of a file (/InFile/) to be passed as an argument, along zero or more execution options. FALTER's response to the options and use of /InFile/ are described in Table 1 below. Should the file named by /InFile/ not exit or not be readable by FALTER, then FALTER shall display the message: File not fond and exit.

Option String	Response
r	!ReadFCFG!
o /OutFile/	//ResultFile// shall be set to /OutFile/
not r	!ReadACFG!
not o	//ResultFile// shall be set to /InFile/
m /Module/	Module named in /Module/ shall be selected for processing
n /NodeID/	Node indicated by /NodeID/ shall be selected as current node

Table 1 -- FALTER Option Processing

3.2.1 !ReadACFG! -- /ACFG/ Input

In the initial execution of FALTER to annotate a particular fault, FALTER shall read in the /ACFG/ generated by REACHER and augment the /ACFGHDR/ and /ABKHDR/ structures to form //FCFGHDR// and //FBKHDR// structures. In each //FBKHDR// in the //FHPROCS// list in //FCFGHDR//, the //FBKFNUM// field shall be set to 0; //FBKFLOC//, //FBKFCON// and //FBKFDATA// all shall be set to an empty list, /ModSelect/ shall be initialized to point to the first //FBKHDR// in //FCFGHDR//. /CurNode/ shall be initialized to point to the first node in //FBKGRPH// and !NewG!. If the **m** and/or **n** options are present, /ModSelect/ and/or /CurNode/, respectively, shall be modified as described in Table 1.

3.2.2 !ReadFCFG! -- //FCFG// Input

To restore a saved //FCFG//, FALTER shall read the file named by /InFile/. The format of this workfile is given in section 3.4. Should the file not be a complete and consistent set of headers and //FCFG// FALTER shall display the message: Invalid workfile format and prompt for an ACFG file to regenerate //FCFG//. Once the data is read in, /ModSelect/ shall be initialized to point to the first //FBKHDR// in //FCFGHDR// and /CurNode/ shall be initialized to point to the first node in //FBKGRPH//. If //FBKFNUM//>0 then using the first elements in //FBKFLOC//, //FBKFCON// and //FBKFDATA//, !OldG!. If //FBKFNUM//=0 then !NewG!. If the **m** and/or **n** options are present, /ModSelect/ and/or /CurNode/, respectively, shall be modified as described in Table 1. If no such //FCFG// exists, FALTER shall display the message: Null workfile and exit.

3.3 //FCFG// Annotation

3.3.1 User Commands

Once an initial //FCFG// is available, either by restoring a previously saved //FCFG// or by augmenting an /ACFG/ constructed by REACHER, FALTER shall allow the user to traverse the //FCFG// and to add to the //FCFG// information on the faults present in the program or program fragment represented by the //FCFG//.

The commands that FALTER shall support to allow the user this functionality are described in table 2, along with a summary of the appropriate response. Supplementary descriptions of the actions required of FALTER in response to these commands are given in the sections that follow. Should the user enter a command that is not listed in table 2, FALTER shall display the message: No such command and prompt the user again. Should the user enter a command listed in table 2 without the listed arguments, FALTER shall display the message: Missing command arguments and prompt the user again, ignoring the partial command. Should the user enter a command with more arguments than those listed in table 2, FALTER shall display the message Ignoring *string* at end of command, where *string* is a list of the extra arguments, and proceed to follow the command, ignoring the extra arguments. Should the user enter a command with arguments that are not of the appropriate type as listed in table 2, FALTER shall display the message: Invalid arguments to command and prompt the user again, ignoring the attempted command. Figure 2 diagrams the FALTER flow of execution through the four classes of commands.

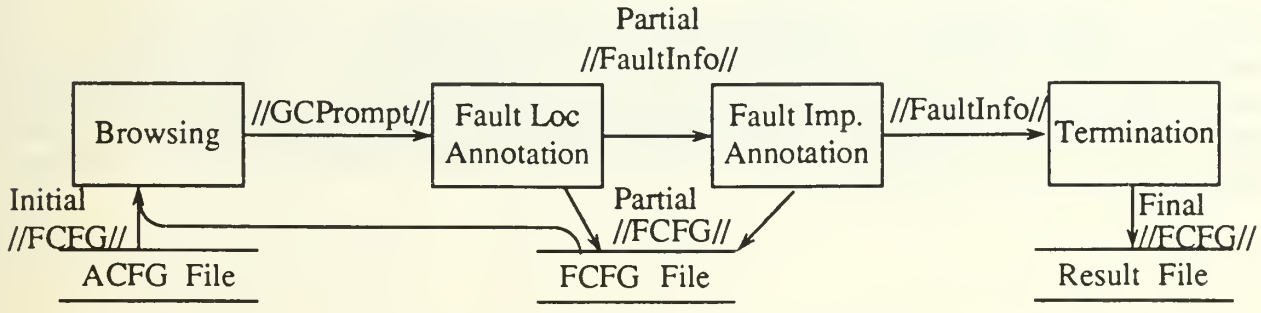

Figure 2 -- FALTER Flow of Execution

Command	Response
a	increment //FBKFNUM// for the current block, !DupF! and using the new entry of //FBKFLOC//, //FBKFCON// and //FBKFDATA//, !OldG!
c /CommStr/	set //GComm// to the value of /CommStr/
e /ErrCond/	set //GErr// to the conditional expressed in /ErrCond/
f /LocCond/	set //FLOC// to /LocCond/
g /ErrNum/	using the entry indicated by /ErrNum/ of //FBKFLOC//, //FBKFCON// and //FBKFDATA//, !OldG!
i /FaultCond/	set //FIMP// to /FaultCond/
m /Module/	set /ModSelect/ to the module named in /Module/
n /NodeID/	set /CurNode/ to the node with ID = /NodeID/
l	set /CurNode/ to the left child of current node
p	set /CurNode/ to the most recently visited node
r	set /CurNode/ to the right child of current node
s	Using the current //GCPrompt// !StoreG!
t /Class/	Set //FTYPE// to the value in //FaultClass//
w /Savefile/	save data structures in the file named in /Savefile/
v /SpecPart/	Set //FVIOL// to the string in /SpecPart/
x	Terminate FALTER execution without saving data structures

Table 2 -- FALTER Command Interpretation

3.3.2 Browsing (**a**, **c**, **e**, **g**, **m**, **n**, **l**, **p**, **r**, **s** commands)

After construction or restoration of the initial //FCFG//, FALTER shall !display! for the appropriate /CurNode/ and prompt the user for a command. The command shall be interpreted as described in table 2.

For the **p**, **l** and **r** commands, FALTER shall not change //GErr// and //GComm//, but FALTER shall vary //GText// and //GCLoc// with the selected /CurNode/. If these commands are entered and there is no previous node, left child or right child (respectively) then FALTER shall display the message Cannot follow arc and prompt for a new command without modification to the data structures..

For the **n** command, if there exists a node in the current module with /ACFGNUM/ equal to the value specified, then FALTER shall not change //GErr// and //GComm//, but FALTER shall vary //GText// and //GCLoc// with the selected /CurNode/. If there does not exist a node in the current module with /ACFGNUM/ equal to the value specified, FALTER shall display the message Node not found and prompt for a new command without modification to the data structures.

For the **m** command, if there exists a module described in //FHPROCS// or its subsidiary //FBKSUBS// entries that has a name equal to the value specified, then FALTER shall !StoreG! and using the new /ModSelect/ !OldG!. If there does not exist such a module description, FALTER shall display the message Module not found and prompt for a new command without modification to any data structures.

For the **a** command, FALTER shall increment //FBKFNUM// and add a new entry in //FBKFLOC//, //FBKFCON// and //FBKFDATA//, duplicating the information from the prior entry, if any. If there is no prior information, then !NewF!.

For the **c** command, //GComm// shall be set to the string given as an argument, with no attempt at validation or format checking of the string.

For the **g** command, if the argument given is in the range 1.../FBKFNUM//, using the //FBKFNUM// for /ModSelect/, then FALTER shall use the designated entry of //FBKFLOC//, //FBKFCON// and //FBKFDATA// and !OldG!, discarding the previous value of //GCPrompt//. If the argument given is 0, then using /ModSelect/ !NewG!. If the argument given is less than 0 or greater than //FBKFNUM// for /ModSelect/ then FALTER shall display the message Value out of range and prompt for a new command without modification of any data structures.

For the **s** command, if //FBKFNUM//=0 then increment //FBKFNUM//, !NewF! and !StoreG!. If //FBKFNUM//>0 then the last entries of //FBKFLOC//, //FBKFCON// and //FBKFDATA// used to set values of //GCPrompt// shall be updated to reflect the current value of //GCPrompt//.

3.3.3 Fault Location Annotation (**f**, **t** commands)

Once a fault is located and informally described, the set of locations that reflect the fault and the precise class of fault located may be annotated in the //FCFG//. The two commands used in this annotation are the **f** and **t** commands.

For the **f** command, if the command argument does not parse to a recognizable /LocCond/ structure then FALTER shall display the message Bad location format and prompt for a new command without modification of any data structures. Otherwise, if //FBKFNUM//>0 then the //FLOC// of the entry of //FBKFDATA// last used to set values of //GCPrompt// shall be updated to the /LocCond/ specified in the command argument. If no such entry exists, then !NewF! and using the new entry FALTER shall update //FLOC// to the /LocCond/ specified in the command argument.

For the **t** command, if the command argument corresponds to one of the defined values for //FTYPE// then FALTER shall replace any old value in //FTYPE// with the value corresponding to the command argument. If the command argument does not correspond to one of the defined values FALTER shall display the message No such fault type and prompt for a new command without modification of any data structures.

3.3.4 Fault Implication Annotation (**i**, **v** commands)

Once the fault is isolated and classified, the implications of the fault in terms of what portion of the specification is violated and what effect the fault has on the system state may be annotated in the //FCFG//. The two commands used in this annotation are the **i** and **v** commands.

For the **i** command, if the command argument does not parse to a recognizable /FaultCond/ structure then FALTER shall display the message Bad implication format and prompt for a new command without modification of any data structures. Otherwise, if //FBKFNUM//>0 then the //FIMP// of the entry of //FBKFDATA// last used to set values of //GCPrompt// shall be updated to the /FaultCond/ specified in the command argument. If no such entry exists or //FBKFNUM//=0, then !NewF! and using the new entry FALTER shall update //FIMP// to the /FaultCond/ specified in the command argument.

For the **v** command, if //FBKFNUM//>0 then the //FVIOL// of the entry of //FBKFDATA// last used to set values of //GCPrompt// shall be set to the string given as an argument, with no attempt at validation or format checking of the string. If no such entry exists or //FBKFNUM//=0, then !NewF! and using the new entry FALTER shall set //FVIOL// to the string given as an argument.

3.3.5 Final Processing (**w**, **x** commands)

Lastly, once the //FCFG// has been appropriately annotated, it may be written out in a form useful for further processing. The precise format described below is intended to be identical to the format expected of SPACER as input.

For the **x** command, FALTER shall request confirmation from the user, and if the command is confirmed, cease execution.

For the **w** command, FALTER shall generate a file recording the //FCFG// in the format used by SPACER as its input language, a LISP structure containing executable analogues of the declarations and statements in the ACFG. The fault annotation will be stored in a structure at the start of the file, with indicators of the apporpriate part of the structure used as location pointers.

4.0 Subsets and Supersets

Supersets

1. Recognition of certain types of faults (i.e., missing logic faults) and specialized handling of those types.
2. Consistency checking employing specialized forms of //FBKFCON//, //FDESC//, and //FVIOL//.
3. Structure to //FVIOL// and //FDESC//

Subsets

1. Less sophisticated handling of fault location.
2. Less sophisticated handling of fault conditions.
3. No **p** command (use **g** as a work-around).

5.0 Undesired Event Handling

Error Messages:

Message	Conditions of generation
Bad implication format	Command argument unrecognizable as fault location
Bad location format	Command argument unrecognizable as fault location
Cannot follow arc	User requested transition along null reference in //FCFG//
File not found	Missing or inaccessible input file.
Ignoring *string* at end of command	Extra arguments on command entered by user.
Invalid arguments to command	Command entered with arguments of wrong type.
Invalid workfile format	Workfile is of wrong format for restoration, or data in workfile is incomplete or inconsistent.
Missing command arguments	Command entered by user without needed arguments.
Module not found	No module in //FHPROCS// or any //FBKSUBS// with //FBKNAME// equal to that specified in the entered command.
No such command	Unrecognized command entered by user.
No such fault type	Unrecognized fault type specified by command argument.
Node not found	No node in current module with /ACFGNUM/ equal to that specified in the entered command.
Null workfile	No //FCFG// nodes in workfile.
Value out of range	Command given with argument with improper value.

6.0 Glossary

!display!	Print the /ACFGNUM/ in //GCLoc//, the //GText// equivalent to the /ACFGTEXT/ in //GCLoc//, and any values set for //GComm// and //GErr//.
!DupF!	If //FBKFNUM//=1 then !NewF!. If //FBKFNUM//>1 then the entries of //FBKFLOC//, //FBKFCON// and //FBKFDATA// corresponding to //FBKFNUM// shall be set to be equal to their immediate predecessors in each list, respectively (i.e., FALTER shall produce a duplicate of the previous fault information in the new entry of these structures).
!NewF!	The new entry of //FBKFLOC// shall be set to /CurNode/; the new entry of //FBKFCON// shall be set to **false**; In the new entry of //FBKFDATA//, //FID// shall be set to /ModSelect/ concatenated with the index of this entry of //FBKFDATA//, //FDESC// and //FVIOL// shall be set to null strings, //FTYPE// shall be set to $Data$, //FLOC// shall be set to the line number corresponding to /CurNode/, //FIMP// shall be set to "**(false) and (false) -> (false)**".
!NewG!	//GCLoc// shall be set to point to /CurNode/, //GText// shall be set to the /ACFGTEXT/ in //GCLoc//, //GComm// shall be set to a null string and //GErr// shall be **false**.
!OldG!	//GCLoc// shall be set to point to the corresponding entry of //FBKFLOC//, //GErr// shall be set to the corresponding entry of //FBKFCON//, //GText// shall be set to the /ACFGTEXT/ in //GCLoc//, //GComm// shall be set to //FDESC// in the corresponding entry of //FBKFDATA//.
!ReadACFG!	See section 3.2.1
!ReadFCFG!	See section 3.2.2
!StoreG!	The corresponding entry of //FBKFLOC// shall be set to //GCLoc//, the corresponding entry of //FBKFCON// shall be set to //GErr//, the corresponding entry of //FBKFDATA// shall be set to have //FDESC// set to //GComm//, and, if //FID// is previously empty, //FID// set to /ModSelect/ concatenated with the index of this entry of //FBKFDATA//.

Distribution List

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 4401 Ford Ave. Alexandria, VA 22302-0268	1
Director of Research Administration Code 012 Naval Postgraduate School Monterey, CA 93943	1
Chairman, Computer Science Department Code 52 Naval Postgraduate School Monterey, CA 93943	1
Shimeall, Timothy J Code 52Sm Naval Postgraduate School Monterey, CA 93943	20